

Package: fastcpd (via r-universe)

October 4, 2026

Type Package

Title Fast Change Point Detection via Sequential Gradient Descent

Version 1.3.1

Description Implements fast change point detection algorithm based on the paper ``Sequential Gradient Descent and Quasi-Newton's Method for Change-Point Analysis" by Xianyang Zhang, Trisha Dawn <<https://proceedings.mlr.press/v206/zhang23b.html>>. The algorithm is based on dynamic programming with pruning and sequential gradient descent. The C++ core is also exposed through a standalone C++ library interface for non-R consumers. See Li and Zhang (2026) <[doi:10.18637/jss.v116.i06](https://doi.org/10.18637/jss.v116.i06)> for details.

License Apache License (>= 2)

URL <https://fastcpd.xingchi.li>, <https://github.com/doccstat/fastcpd>

BugReports <https://github.com/doccstat/fastcpd/issues>

Depends R (>= 2.10)

Imports methods, Rcpp (>= 0.11.0), stats

Suggests ggplot2, gridExtra, knitr, matrixStats, mvtnorm, RcppArmadillo, rmarkdown, testthat (>= 3.0.0), xml2

LinkingTo abseil, Rcpp, RcppArmadillo, testthat

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first examples-fastcpd_arima

Encoding UTF-8

LazyData true

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make libuv1-dev

Repository <https://doccstat.r-universe.dev>

Date/Publication 2026-10-04 20:32:37 UTC

RemoteUrl <https://github.com/doccstat/fastcpd>

RemoteRef HEAD

RemoteSha 16c52a8ce259f971a028946e741f406d4c296f24

Contents

bitcoin	3
confint.fastcpd	4
detect	6
detect_ar	16
detect_arima	17
detect_arma	18
detect_binomial	20
detect_exponential	21
detect_family	22
detect_garch	22
detect_kernel	24
detect_lasso	25
detect_lm	27
detect_mean	28
detect_meanvariance	29
detect_poisson	31
detect_quantile	32
detect_rank	33
detect_var	34
detect_variance	35
estimate_variance	36
estimate_variance_arma	37
estimate_variance_linear_regression	38
estimate_variance_mean	39
estimate_variance_median	40
fastcpd-class	41
occupancy	42
plot.fastcpd	42
plot.fastcpd_confint	45
print.fastcpd	46
show.fastcpd	47
summary.fastcpd	47
transcriptome	48
uk_seatbelts	50
well_log	52

Index

54

bitcoin	<i>Bitcoin Market Price (USD)</i>
---------	-----------------------------------

Description

The average USD market price across major bitcoin exchanges.

Usage

```
bitcoin
```

Format

A data frame with 1354 rows and 2 variables:

date POSIXct, POSIXt (TZ: "UTC") from 2019-01-02 to 2023-10-28

price The average USD market price across major bitcoin exchanges

Source

<<https://www.blockchain.com/explorer/charts/market-price>>

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  p <- ggplot2::ggplot(bitcoin, ggplot2::aes(x = date, y = price)) +
    ggplot2::geom_line()
  print(p)

  result <- suppressWarnings(detect_garch(
    diff(log(bitcoin$price[600:900])), c(1, 1),
    beta = "BIC", cost_adjustment = "BIC"
  ))
  summary(result)
  bitcoin$date[result@cp_set + 600]
  plot(result)

  cp_dates <- bitcoin[600 + result@cp_set + 1, "date"]
  ggplot2::ggplot(
    data = data.frame(
      x = bitcoin$date[600:900], y = bitcoin$price[600:900]
    ),
    ggplot2::aes(x = x, y = y)
  ) +
    ggplot2::geom_line(color = "steelblue") +
    ggplot2::geom_vline(
      xintercept = cp_dates,
      color = "red",
      linetype = "dotted",
```

```

    linewidth = 0.5,
    alpha = 0.7
  ) +
  ggplot2::labs(
    x = "Year",
    y = "Bitcoin price in USD"
  ) +
  ggplot2::annotate(
    "text",
    x = cp_dates,
    y = 2000,
    label = as.character(cp_dates),
    color = "steelblue"
  ) +
  ggplot2::theme_bw()
}

```

confint.fastcpd

Confidence intervals for a fastcpd object

Description

Confidence intervals for a fastcpd object

Usage

```

## S3 method for class 'fastcpd'
confint(
  object,
  parm = c("cp", "theta"),
  level = 0.95,
  method = NULL,
  B = 999,
  bootstrap = c("nonparametric"),
  window = NULL,
  min_segment_length = 2L,
  seed = NULL,
  refit_envir = parent.frame(),
  ...
)

```

Arguments

object	A fastcpd object.
parm	The target parameter. Use "cp" for change-point locations or "theta" for segment parameters.
level	Confidence level.

method	Method used to construct intervals. Change-point intervals support "bootstrap" and "profile". Parameter intervals support "wald"; multivariate linear-model Wald intervals are not currently part of the portable R/Python contract.
B	Number of bootstrap replicates when method = "bootstrap".
bootstrap	Bootstrap type. Currently "nonparametric" resamples observations within each estimated segment and is available for all families that can be refitted from object@call.
window	Optional half-width around each detected change point for profile intervals. If NULL, the whole interval between neighboring detected change points is profiled.
min_segment_length	Minimum number of observations on each side of a candidate split for profile intervals.
seed	Optional random seed used for bootstrap reproducibility.
refit_envir	Environment used to evaluate bootstrap refits and arguments stored in object@call, such as variance_estimation.
...	Ignored.

Value

A data frame of class `fastcpd_confint` containing estimates, lower and upper interval bounds, and method-specific diagnostics. The returned intervals can be visualized with `plot.fastcpd_confint()`.

Examples

```
# The well-log data contain outliers, so a robust variance estimate keeps
# the detection and the confidence intervals stable.
sigma2 <- estimate_variance_median(well_log)
result <- detect_mean(well_log, trim = 0.001, variance_estimation = sigma2)

(cp_profile_interval <- confint(
  result,
  parm = "cp",
  method = "profile",
  level = 0.8,
  window = 8
))

(theta_wald_interval <- confint(
  result,
  parm = "theta",
  method = "wald"
))

(cp_bootstrap_interval <- confint(
  result,
  parm = "cp",
  method = "bootstrap",
  B = 5,
  seed = 10
))
```

```

))

variance_result <- detect_variance(well_log)
# Wald intervals are not available for variance-family fits.

# ARIMA profile intervals use the same segment-local native likelihood as the
# detector rather than a separate stats::arima refit.
arima_small <- rep(c(0.1, -0.1), 20)
arima_large <- rep(c(2, -2), length.out = 41)
arima_data <- c(0, cumsum(c(arima_small, arima_large)))
arima_result <- detect_arima(
  arima_data,
  order = c(0, 1, 0)
)
arima_profile_interval <- confint(
  arima_result,
  parm = "cp",
  method = "profile",
  level = 0.8,
  window = 1
)

```

detect

Find change points efficiently

Description

`detect()` takes in formulas, data, families and extra parameters and returns a [fastcpd](#) object.

Usage

```

detect(
  formula = y ~ . - 1,
  data,
  beta = "MBIC",
  cost_adjustment = "MBIC",
  family = NULL,
  cost = NULL,
  cost_gradient = NULL,
  cost_hessian = NULL,
  line_search = c(1),
  lower = rep(-Inf, p),
  upper = rep(Inf, p),
  pruning_coef = 0,
  segment_count = 10,
  trim = 0,
  momentum_coef = 0,
  multiple_epochs = function(x) 0,
  epsilon = 1e-10,

```

```

    order = NULL,
    p = ncol(data) - 1,
    variance_estimation = NULL,
    cp_only = FALSE,
    vanilla_percentage = 0,
    warm_start = FALSE,
    ...
)

fastcpd(
  formula = y ~ . - 1,
  data,
  beta = "MBIC",
  cost_adjustment = "MBIC",
  family = NULL,
  cost = NULL,
  cost_gradient = NULL,
  cost_hessian = NULL,
  line_search = c(1),
  lower = rep(-Inf, p),
  upper = rep(Inf, p),
  pruning_coef = 0,
  segment_count = 10,
  trim = 0,
  momentum_coef = 0,
  multiple_epochs = function(x) 0,
  epsilon = 1e-10,
  order = NULL,
  p = ncol(data) - 1,
  variance_estimation = NULL,
  cp_only = FALSE,
  vanilla_percentage = 0,
  warm_start = FALSE,
  ...
)

```

Arguments

formula	A formula object specifying the model to be fitted. The (optional) response variable should be on the LHS of the formula, while the covariates should be on the RHS. The naming of variables used in the formula should be consistent with the column names in the data frame provided in data. The intercept term should be removed from the formula. The response variable is not needed for mean/variance change models and time series models. By default, an intercept column will be added to the data, similar to the <code>lm()</code> function. Thus, it is suggested that users should remove the intercept term by appending <code>- 1</code> to the formula. Note that the <code>detect_family</code> functions do not require a formula input.
data	A data frame of dimension $T \times d$ containing the data to be segmented (where

each row denotes a data point $z_t \in \mathbb{R}^d$ for $t = 1, \dots, T$) is required in the main function, while a matrix or a vector input is also accepted in the `detect_family` functions.

beta	Penalty criterion for the number of change points. This parameter takes a string value of "BIC", "MBIC", "MDL" or a numeric value. If a numeric value is provided, the value will be used as the penalty. By default, the mBIC criterion is used, where $\beta = (p + 2) \log(T)/2$. This parameter usage should be paired with <code>cost_adjustment</code> described below. Discussions about the penalty criterion can be found in the references.
cost_adjustment	Cost adjustment criterion. It can be "BIC", "MBIC", "MDL" or NULL. By default, the cost adjustment criterion is set to be "MBIC". The "MBIC" and "MDL" criteria modify the cost function by adding a negative adjustment term to the cost function. "BIC" or NULL does not modify the cost function. Details can in found in the references.
family	Family class of the change point model. It can be "mean" for mean change, "variance" for variance change, "meanvariance" for mean and/or variance change, "lm" for linear regression, "binomial" for logistic regression, "poisson" for Poisson regression, "lasso" for penalized linear regression, "ar" for AR(p) models, "arma" for ARMA(p, q) models, "arima" for ARIMA(p, d, q) models, "garch" for GARCH(p, q) models, "var" for VAR(p) models and "custom" for user-specified custom models. Omitting this parameter is the same as specifying the parameter to be "custom" or NULL, in which case, users must specify the custom cost function.
cost	Cost function to be used. <code>cost</code> , <code>cost_gradient</code> , and <code>cost_hessian</code> should not be specified at the same time with <code>family</code> as built-in families have cost functions implemented in C++ to provide better performance. If not specified, the default is the negative log-likelihood for the corresponding family. Custom cost functions can be provided in the following two formats:

- `cost = function(data) { ... }`
- `cost = function(data, theta) { ... }`

Users can specify a loss function using the second format that will be used to calculate the cost value. In both formats, the input data is a subset of the original data frame in the form of a matrix (a matrix with a single column in the case of a univariate data set). In the first format, the specified cost function directly calculates the cost value. `detect()` performs the vanilla PELT algorithm, and `cost_gradient` and `cost_hessian` should not be provided since no parameter updating is necessary for vanilla PELT. In the second format, the loss function $\sum_{i=s}^t l(z_i, \theta)$ is provided, which has to be optimized over the parameter θ to obtain the cost value. A detailed discussion about the custom cost function usage can be found in the references.

Compiled cost functions. For performance-sensitive use cases, `cost` (as well as `cost_gradient` and `cost_hessian`) may instead be a pre-compiled C++ function passed as an external pointer (`externalptr`), avoiding R-call overhead in the hot loop. Build one with `Rcpp::XPtr`: write a function matching one of `double cost(arma::mat const& data)` (PELT-style, format one above) or

double cost(arma::mat const& data, arma::colvec const& theta) (SeGD-style, format two above), take its address, and wrap it as `xptr <- Rcpp::XPtr<FnPtr>(new FnPtr(&your_cost), TRUE, Rcpp::wrap("fastcpd_cost_pelt"))` (or "fastcpd_cost_sen" for the two-argument form) – the tag string is required and checked at runtime. Since external pointers carry no formals, also set `attr(xptr, "fastcpd_cost_arity") <- 1L` (or 2L for the two-argument form) so `detect()` can route it like an R closure of the same arity. Pass `xptr` as `cost` exactly as you would an R closure. A compiled `cost` cannot be combined with `cost_gradient` / `cost_hessian` (those drive an R-level `stats::optim` warm start that requires an R closure for `cost`). See the custom model vignette for a complete worked example.

`cost_gradient` Gradient of the custom cost function. Example usage:

```
cost_gradient = function(data, theta) {
  ...
  return(gradient)
}
```

The gradient function takes two inputs, the first being a matrix representing a segment of the data, similar to the format used in the `cost` function, and the second being the parameter that needs to be optimized. The gradient function returns the value of the gradient of the loss function, i.e., $\sum_{i=s}^t \nabla l(z_i, \theta)$. Like `cost`, this may also be a compiled function passed as an externalptr matching `arma::colvec cost_gradient(arma::mat const& data, arma::colvec const& theta)`, wrapped via `Rcpp::XPtr` and tagged "fastcpd_cost_gradient".

`cost_hessian` Hessian of the custom loss function. The Hessian function takes two inputs, the first being a matrix representing a segment of the data, similar to the format used in the `cost` function, and the second being the parameter that needs to be optimized. The gradient function returns the Hessian of the loss function, i.e., $\sum_{i=s}^t \nabla^2 l(z_i, \theta)$. Like `cost`, this may also be a compiled function passed as an externalptr matching `arma::mat cost_hessian(arma::mat const& data, arma::colvec const& theta)`, wrapped via `Rcpp::XPtr` and tagged "fastcpd_cost_hessian".

`line_search` If a vector of numeric values is provided, a line search will be performed to find the optimal step size for each update. Detailed usage of `line_search` can be found in the references.

`lower` Lower bound for the parameters. Used to specify the domain of the parameters after each gradient descent step. If not specified, the lower bound is set to be $-\text{Inf}$ for all parameters. `lower` is especially useful when the estimated parameters take only positive values, such as the noise variance.

`upper` Upper bound for the parameters. Used to specify the domain of the parameters after each gradient descent step. If not specified, the upper bound is set to be Inf for all parameters.

`pruning_coef` Pruning coefficient `$c_0$` used in the pruning step of the PELT algorithm with the default value 0. If `cost_adjustment` is specified as "MBIC", an adjustment term $p \log(2)$ will be added to the pruning coefficient. If `cost_adjustment` is specified as "MDL", an adjustment term $p \log_2(2)$ will be added to the pruning coefficient. Detailed discussion about the pruning coefficient can be found in the references.

segment_count	An initial guess of the number of segments. If not specified, the initial guess of the number of segments is 10. The initial guess affects the initial estimates of the parameters in SeGD.
trim	Trimming for the boundary change points so that a change point close to the boundary will not be counted as a change point. This parameter also specifies the minimum distance between two change points. If several change points have mutual distances smaller than $\text{trim} * \text{nrow}(\text{data})$, those change points will be merged into one single change point. The value of this parameter should be between 0 and 1.
momentum_coef	Momentum coefficient to be applied to each update. This parameter is used when the loss function is bad-shaped so that maintaining a momentum from previous update is desired. Default value is 0, meaning the algorithm doesn't maintain a momentum by default.
multiple_epochs	A function can be specified such that an adaptive number of multiple epochs can be utilized to improve the algorithm's performance. <code>multiple_epochs</code> is a function of the length of the data segment. The function returns an integer indicating how many epochs should be performed apart from the default update. By default, the function returns zero, meaning no multiple epochs will be used to update the parameters. Example usage: <pre>multiple_epochs = function(segment_length) { if (segment_length < 100) 1 else 0 }</pre> This function will let SeGD perform parameter updates with an additional epoch for each segment with a length less than 100 and no additional epoch for segments with lengths greater or equal to 100. Callable schedules are native R and standalone C++ extensions outside the portable built-in numerical contract. Python intentionally keeps detection GIL-free and accepts only <code>multiple_epochs = NULL</code>.
epsilon	Epsilon to avoid numerical issues. Only used for the Hessian computation in Logistic Regression and Poisson Regression.
order	Order of the $\text{AR}(p)$, $\text{VAR}(p)$ or $\text{ARIMA}(p, d, q)$ model. If omitted, a family-specific default is used ($\text{AR}(1)$, $\text{ARMA}(1, 1)$, $\text{ARIMA}(1, 1, 0)$, $\text{GARCH}(1, 1)$, $\text{VAR}(1)$, median quantile, or 100-feature KCP).
p	Number of covariates in the model. If not specified, the number of covariates will be inferred from the data, i.e., $p = \text{ncol}(\text{data}) - 1$. This parameter is superseded by <code>order</code> in the case of time series models: "ar", "var", "arima".
variance_estimation	An estimate of the variance / covariance matrix for the data. If not specified, the variance / covariance matrix will be estimated using the data.
cp_only	If TRUE, only the change points are returned. Otherwise, the cost function values together with the estimated parameters for each segment are also returned. By default the value is set to be FALSE so that <code>plot</code> can be used to visualize the results for a built-in model. <code>cp_only</code> has some performance impact on the algorithm, since the cost values and estimated parameters for each segment need

to be calculated and stored. If the users are only interested in the change points, setting `cp_only` to be `TRUE` will help with the computational cost.

`vanilla_percentage`

The parameter v is between zero and one. For each segment, when its length is no more than vT , the cost value will be computed by performing an exact minimization of the loss function over the parameter. When its length is greater than vT , the cost value is approximated through SeGD. Therefore, this parameter induces an algorithm that can be interpreted as an interpolation between dynamic programming with SeGD ($v = 0$) and the vanilla PELT ($v = 1$). The readers are referred to the references for more details.

`warm_start`

If `TRUE`, the algorithm will use the estimated parameters from the previous segment as the initial value for the current segment. This parameter is only used for the "glm" families.

`...`

Other parameters for specific models.

- `include.mean` is retained for call compatibility in ARIMA models. It defaults to `FALSE`; `TRUE` is unsupported because R and Python share a zero-mean native ARMA likelihood.
- `show.progress` is used to control the progress bar. By default no progress bar is shown. Set `show.progress = TRUE` to display a tqdm-format progress bar on `stderr` showing PELT timestep progress.
- `p.response` is used to specify the number of response variables. This parameter is especially useful for linear models with multivariate responses.

Value

A `fastcpd` object.

Gallery

<https://github.com/doccstat/fastcpd/tree/r/tests/testthat/examples>

References

Xingchi Li, Xianyang Zhang (2026). "fastcpd: Fast Change Point Detection in R." *Journal of Statistical Software*, **116**(6), 1–53. doi:10.18637/jss.v116.i06.

Xingchi Li, Xianyang Zhang (2024). "fastcpd: Fast Change Point Detection in R." *arXiv:2404.05933*, <https://arxiv.org/abs/2404.05933>.

Xianyang Zhang, Trisha Dawn (2023). "Sequential Gradient Descent and Quasi-Newton's Method for Change-Point Analysis." In Ruiz, Francisco, Dy, Jennifer, van de Meent, Jan-Willem (eds.), *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 series Proceedings of Machine Learning Research, 1129–1143.

See Also

`detect_family` for the family-specific function; `plot.fastcpd()` for plotting the results, `summary.fastcpd()` for summarizing the results.

Examples

```

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 200
  p <- 4
  d <- 2
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_1 <- matrix(runif(8, -3, -1), nrow = p)
  theta_2 <- matrix(runif(8, -1, 3), nrow = p)
  y <- rbind(
    x[1:125, ] %%% theta_1 + mvtnorm::rmvnorm(125, rep(0, d), 3 * diag(d)),
    x[126:n, ] %%% theta_2 + mvtnorm::rmvnorm(75, rep(0, d), 3 * diag(d))
  )
  result_mlm <- detect(
    cbind(y.1, y.2) ~ . - 1, cbind.data.frame(y = y, x = x), family = "lm"
  )
  summary(result_mlm)
}
if (
  requireNamespace("mvtnorm", quietly = TRUE) &&
  requireNamespace("stats", quietly = TRUE)
) {
  set.seed(1)
  n <- 400 + 300 + 500
  p <- 5
  x <- mvtnorm::rmvnorm(n, mean = rep(0, p), sigma = diag(p))
  theta <- rbind(
    mvtnorm::rmvnorm(1, mean = rep(0, p - 3), sigma = diag(p - 3)),
    mvtnorm::rmvnorm(1, mean = rep(5, p - 3), sigma = diag(p - 3)),
    mvtnorm::rmvnorm(1, mean = rep(9, p - 3), sigma = diag(p - 3))
  )
  theta <- cbind(theta, matrix(0, 3, 3))
  theta <- theta[rep(seq_len(3), c(400, 300, 500)), ]
  y_true <- rowSums(x * theta)
  factor <- c(
    2 * stats::rbinom(400, size = 1, prob = 0.95) - 1,
    2 * stats::rbinom(300, size = 1, prob = 0.95) - 1,
    2 * stats::rbinom(500, size = 1, prob = 0.95) - 1
  )
  y <- factor * y_true + stats::rnorm(n)
  data <- cbind.data.frame(y, x)
  huber_threshold <- 1
  huber_loss <- function(data, theta) {
    residual <- data[, 1] - data[, -1, drop = FALSE] %%% theta
    indicator <- abs(residual) <= huber_threshold
    sum(
      residual^2 / 2 * indicator +
        huber_threshold * (
          abs(residual) - huber_threshold / 2
        ) * (1 - indicator)
    )
  }
}

```

```

huber_loss_gradient <- function(data, theta) {
  residual <- c(data[nrow(data), 1] - data[nrow(data), -1] %*% theta)
  if (abs(residual) <= huber_threshold) {
    -residual * data[nrow(data), -1]
  } else {
    -huber_threshold * sign(residual) * data[nrow(data), -1]
  }
}
huber_loss_hessian <- function(data, theta) {
  residual <- c(data[nrow(data), 1] - data[nrow(data), -1] %*% theta)
  if (abs(residual) <= huber_threshold) {
    outer(data[nrow(data), -1], data[nrow(data), -1])
  } else {
    0.01 * diag(length(theta))
  }
}
huber_regression_result <- detect(
  formula = y ~ . - 1,
  data = data,
  beta = (p + 1) * log(n) / 2,
  cost = huber_loss,
  cost_gradient = huber_loss_gradient,
  cost_hessian = huber_loss_hessian,
  trim = 0.05
)
summary(huber_regression_result)
}

set.seed(1)
p <- 1
x <- matrix(rnorm(375 * p, 0, 1), ncol = p)
theta <- rbind(rnorm(p, 0, 1), rnorm(p, 2, 1))
y <- c(
  rbinom(200, 1, 1 / (1 + exp(-x[1:200, ] %*% theta[1, , drop = FALSE]))),
  rbinom(175, 1, 1 / (1 + exp(-x[201:375, ] %*% theta[2, , drop = FALSE]))))
)
data <- data.frame(y = y, x = x)
result_builtin <- suppressWarnings(detect_binomial(data))
logistic_loss <- function(data, theta) {
  x <- data[, -1, drop = FALSE]
  y <- data[, 1]
  u <- x %*% theta
  nll <- -y * u + log(1 + exp(u))
  nll[u > 10] <- -y[u > 10] * u[u > 10] + u[u > 10]
  sum(nll)
}
logistic_loss_gradient <- function(data, theta) {
  x <- data[nrow(data), -1, drop = FALSE]
  y <- data[nrow(data), 1]
  c(-(y - 1 / (1 + exp(-x %*% theta)))) * x
}
logistic_loss_hessian <- function(data, theta) {
  x <- data[nrow(data), -1]

```

```

    prob <- 1 / (1 + exp(-x %*% theta))
    (x %*% x) * c((1 - prob) * prob)
  }
result_custom <- detect(
  formula = y ~ . - 1,
  data = data,
  epsilon = 1e-5,
  cost = logistic_loss,
  cost_gradient = logistic_loss_gradient,
  cost_hessian = logistic_loss_hessian
)
result_builtincp_set
result_customcp_set

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 480
  p_true <- 6
  p <- 50
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_0 <- rbind(
    runif(p_true, -5, -2),
    runif(p_true, -3, 3),
    runif(p_true, 2, 5),
    runif(p_true, -5, 5)
  )
  theta_0 <- cbind(theta_0, matrix(0, ncol = p - p_true, nrow = 4))
  y <- c(
    x[1:80, ] %*% theta_0[1, ] + rnorm(80, 0, 1),
    x[81:200, ] %*% theta_0[2, ] + rnorm(120, 0, 1),
    x[201:320, ] %*% theta_0[3, ] + rnorm(120, 0, 1),
    x[321:n, ] %*% theta_0[4, ] + rnorm(160, 0, 1)
  )
  small_lasso_data <- cbind.data.frame(y, x)
  result_no_vp <- detect_lasso(
    small_lasso_data,
    beta = "BIC",
    cost_adjustment = NULL,
    pruning_coef = 0
  )
  summary(result_no_vp)
  result_20_vp <- detect_lasso(
    small_lasso_data,
    beta = "BIC",
    cost_adjustment = NULL,
    vanilla_percentage = 0.2,
    pruning_coef = 0
  )
  summary(result_20_vp)
}

```

```

if (requireNamespace("RcppArmadillo", quietly = TRUE)) {
  # Three equivalent mean-change detectors: built-in "mean" family (fully
  # compiled), a custom R closure, and a compiled Rcpp::XPtr cost function.
  # All three detect the same change point; the XPtr eliminates the per-
  # candidate R call overhead that the R closure incurs in the hot loop.
  set.seed(1)
  data <- matrix(c(rnorm(500, 0), rnorm(500, 5)))
  beta_val <- (ncol(data) + 1) * log(nrow(data)) / 2

  result_builtin <- detect_mean(
    data, beta = beta_val, cost_adjustment = NULL
  )

  cost_r <- function(data) sum((data - mean(data))^2) / 2
  time_r <- system.time(
    result_r <- detect(
      ~ . - 1, data.frame(x = data), family = "custom",
      cost = cost_r, beta = beta_val, cost_adjustment = NULL
    )
  )

  compiled_env <- new.env()
  Rcpp::sourceCpp(
    code = '
      // [[Rcpp::depends(RcppArmadillo)]]
      #include <RcppArmadillo.h>
      // PELT-style cost: one argument, data only (no theta).
      // Tag the XPtr with "fastcpd_cost_pelt" and set
      // attr(xptr, "fastcpd_cost_arity") <- 1L on the R side.
      typedef double (*CostPeltFnPtr)(arma::mat const&);

      double mean_cost(arma::mat const& data) {
        arma::rowvec const seg_mean = arma::mean(data, 0);
        return arma::accu(arma::square(data.each_row() - seg_mean)) / 2.0;
      }

      // [[Rcpp::export]]
      SEXP make_mean_cost_xptr() {
        Rcpp::XPtr<CostPeltFnPtr> xptr(
          new CostPeltFnPtr(&mean_cost), true,
          Rcpp::wrap("fastcpd_cost_pelt"));
        return xptr;
      }
    ',
    env = compiled_env
  )

  cost_xptr <- compiled_env$make_mean_cost_xptr()
  attr(cost_xptr, "fastcpd_cost_arity") <- 1L
  time_xptr <- system.time(
    result_xptr <- detect(
      ~ . - 1, data.frame(x = data), family = "custom",
      cost = cost_xptr, beta = beta_val, cost_adjustment = NULL
    )
  )

```

```

)

stopifnot(
  identical(result_r@cp_set, result_builtin@cp_set),
  identical(result_xptr@cp_set, result_builtin@cp_set)
)
cat("R closure: ", time_r["elapsed"], "s\n")
cat("XPtr:      ", time_xptr["elapsed"], "s\n")
summary(result_xptr)
}

```

detect_ar

*Find change points efficiently in AR(p) models***Description**

[detect_ar\(\)](#) and [fastcpd.ar\(\)](#) are wrapper functions of [detect\(\)](#) to find change points in AR(p) models. The function is similar to [detect\(\)](#) except that the data is by default a one-column matrix or univariate vector and thus a formula is not required here.

Usage

```
detect_ar(data, order = 1, ...)
```

```
fastcpd_ar(data, order = 1, ...)
```

```
fastcpd.ar(data, order = 1, ...)
```

Arguments

data	A numeric vector, a matrix, a data frame or a time series object.
order	A positive integer specifying the order of the AR model; defaults to 1 (AR(1)).
...	Other arguments passed to detect() , for example, <code>segment_count</code> .

Value

A [fastcpd](#) object.

See Also

[detect\(\)](#)

Examples

```

set.seed(1)
n <- 1000
x <- rep(0, n + 3)
for (i in 1:600) {
  x[i + 3] <- 0.6 * x[i + 2] - 0.2 * x[i + 1] + 0.1 * x[i] + rnorm(1, 0, 3)
}
for (i in 601:1000) {
  x[i + 3] <- 0.3 * x[i + 2] + 0.4 * x[i + 1] + 0.2 * x[i] + rnorm(1, 0, 3)
}
result <- detect_ar(x[3 + seq_len(n)], 3)
summary(result)
plot(result)
set.seed(1)
n <- 1000
x <- rep(0, n + 3)
for (i in 1:600) {
  x[i + 3] <- 0.6 * x[i + 2] - 0.2 * x[i + 1] + 0.1 * x[i] + rnorm(1, 0, 3)
}
for (i in 601:1000) {
  x[i + 3] <- 0.3 * x[i + 2] + 0.4 * x[i + 1] + 0.2 * x[i] + rnorm(1, 0, 3)
}
result <-
  detect_ar(x[3 + seq_len(n)], 3, beta = "MDL", cost_adjustment = "MDL")
summary(result)
plot(result)

```

detect_arima

Find change points efficiently in ARIMA(p , d , q) models

Description

`detect_arima()` and `fastcpd.arima()` are wrapper functions of `detect()` to find change points in ARIMA(p , d , q) models. Differencing is performed independently inside each candidate segment, so no artificial difference is formed across a proposed change-point boundary. Change-point indices refer to the original, undifferenced series. When $d = 0$, this is the same model as `detect_arma()`. The function is similar to `detect()` except that the data is by default a one-column matrix or univariate vector and thus a formula is not required here.

Usage

```

detect_arima(data, order = c(1, 1, 0), ...)

fastcpd_arima(data, order = c(1, 1, 0), ...)

fastcpd.arima(data, order = c(1, 1, 0), ...)

```

Arguments

data	A numeric vector, a matrix, a data frame or a time series object.
order	A vector of length three specifying the order of the ARIMA model; defaults to <code>c(1, 1, 0)</code> .
...	Other arguments passed to <code>detect()</code> , for example, <code>segment_count</code> . The ARIMA-specific <code>include.mean</code> option is accepted here, defaults to <code>FALSE</code> , and must remain <code>FALSE</code> .

Value

A `fastcpd` object.

See Also

`detect()`

Examples

```
small_increments <- rep(c(0.1, -0.1), 20)
large_increments <- rep(c(2, -2), length.out = 41)
x <- c(0, cumsum(c(small_increments, large_increments)))
result <- detect_arma(
  x,
  order = c(0, 1, 0)
)
summary(result)
plot(result)
```

detect_arma

Find change points efficiently in ARMA(p , q) models

Description

`detect_arma()` and `fastcpd.arma()` are wrapper functions of `detect()` to find change points in ARMA(p , q) models. The function is similar to `detect()` except that the data is by default a one-column matrix or univariate vector and thus a formula is not required here. Pure AR models ($q = 0$) use the same lagged Gaussian path as `detect_ar()`; pure MA models ($p = 0$) use the native MA family.

Usage

```
detect_arma(data, order = c(1, 1), ...)
```

```
fastcpd_arma(data, order = c(1, 1), ...)
```

```
fastcpd.arma(data, order = c(1, 1), ...)
```

Arguments

data	A numeric vector, a matrix, a data frame or a time series object.
order	A vector of length two specifying the order of the ARMA model; defaults to <code>c(1, 1)</code> .
...	Other arguments passed to <code>detect()</code> , for example, <code>segment_count</code> .

Value

A `fastcpd` object.

See Also

`detect()`

Examples

```
set.seed(1)
n <- 200
w <- rnorm(n + 3, 0, 3)
x <- rep(0, n + 3)
for (i in 1:150) {
  x[i + 3] <- 0.1 * x[i + 2] - 0.3 * x[i + 1] + 0.1 * x[i] +
    0.1 * w[i + 2] + 0.5 * w[i + 1] + w[i + 3]
}
for (i in 151:n) {
  x[i + 3] <- 0.3 * x[i + 2] + 0.1 * x[i + 1] - 0.3 * x[i] -
    0.6 * w[i + 2] - 0.1 * w[i + 1] + w[i + 3]
}
result <- suppressWarnings(
  detect_arma(
    data = x[3 + seq_len(n)],
    order = c(3, 2),
    segment_count = 3,
    lower = c(rep(-1, 3 + 2), 1e-10),
    upper = c(rep(1, 3 + 2), Inf),
    line_search = c(1, 0.1, 1e-2),
    beta = "BIC",
    cost_adjustment = "BIC",
    trim = 0.025
  )
)
summary(result)
plot(result)
```

detect_binomial	<i>Find change points efficiently in logistic regression models</i>
-----------------	---

Description

`detect_binomial()` and `fastcpd.binomial()` are wrapper functions of `detect()` to find change points in logistic regression models. The function is similar to `detect()` except that the data is by default a matrix or data frame with the response variable as the first column and thus a formula is not required here.

Usage

```
detect_binomial(data, ...)

detect_logistic_regression(data, ...)

fastcpd_binomial(data, ...)

fastcpd.binomial(data, ...)
```

Arguments

<code>data</code>	A matrix or a data frame with the response variable as the first column.
<code>...</code>	Other arguments passed to <code>detect()</code> , for example, <code>segment_count</code> .

Value

A `fastcpd` object.

See Also

[detect\(\)](#)

Examples

```
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 500
  p <- 4
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta <- rbind(rnorm(p, 0, 1), rnorm(p, 2, 1))
  y <- c(
    rbinom(300, 1, 1 / (1 + exp(-x[1:300, ] %*% theta[1, ]))),
    rbinom(200, 1, 1 / (1 + exp(-x[301:n, ] %*% theta[2, ])))
  )
  result <- suppressWarnings(detect_binomial(cbind(y, x)))
  summary(result)
  plot(result)
}
```

detect_exponential	<i>Find change points efficiently in exponentially distributed data</i>
--------------------	---

Description

`detect_exponential()` and `fastcpd.exponential()` are wrapper functions of `detect()` to find changes in the rate of exponentially distributed data, i.e. mean change under exponentially distributed noise (cf. `changepoint::cpt.meanvar` with `test.stat = "Exponential"`). The function is similar to `detect()` except that the data is by default a matrix or data frame or a vector with each row / element as an observation and thus a formula is not required here.

Usage

```
detect_exponential(data, ...)  
  
fastcpd_exponential(data, ...)  
  
fastcpd.exponential(data, ...)
```

Arguments

<code>data</code>	A matrix, a data frame or a vector.
<code>...</code>	Other arguments passed to <code>detect()</code> , for example, <code>segment_count</code> .

Value

A `fastcpd` object.

See Also

[detect\(\)](#)

Examples

```
set.seed(1)  
data <- matrix(c(  
  rexp(300, rate = 1),  
  rexp(400, rate = 0.1),  
  rexp(300, rate = 1)  
)  
)  
system.time(result <- detect_exponential(data))  
summary(result)  
plot(result)
```

detect_family	<i>Family-specific change point detection</i>
---------------	---

Description

Convenience functions built on [detect\(\)](#) for common models.

See Also

Basic statistics: [detect_mean\(\)](#), [detect_variance\(\)](#), [detect_meanvariance\(\)](#), [detect_exponential\(\)](#);
 Regression: [detect_lm\(\)](#), [detect_binomial\(\)](#), [detect_poisson\(\)](#), [detect_lasso\(\)](#), [detect_quantile\(\)](#);
 Time series: [detect_ar\(\)](#), [detect_var\(\)](#), [detect_arima\(\)](#), [detect_arma\(\)](#), [detect_garch\(\)](#);
 Distribution-free: [detect_kernel\(\)](#), [detect_rank\(\)](#).

detect_garch	<i>Find change points efficiently in GARCH(p, q) models</i>
--------------	---

Description

[detect_garch\(\)](#) and [fastcpd.garch\(\)](#) are wrapper functions of [detect\(\)](#) to find change points in GARCH(p , q) models. The function is similar to [detect\(\)](#) except that the data is by default a one-column matrix or univariate vector and thus a formula is not required here.

Usage

```
detect_garch(data, order = c(1, 1), ...)  
  
fastcpd_garch(data, order = c(1, 1), ...)  
  
fastcpd.garch(data, order = c(1, 1), ...)
```

Arguments

data	A numeric vector, a matrix, a data frame or a time series object.
order	A positive integer vector of length two specifying the order of the GARCH model; defaults to c(1, 1).
...	Other arguments passed to detect() , for example, segment_count.

Value

A [fastcpd](#) object.

See Also

[detect\(\)](#)

Examples

```

set.seed(1)
n <- 401
sigma_2 <- rep(1, n + 1)
x <- rep(0, n + 1)
for (i in seq_len(200)) {
  sigma_2[i + 1] <- 20 + 0.8 * x[i]^2 + 0.1 * sigma_2[i]
  x[i + 1] <- rnorm(1, 0, sqrt(sigma_2[i + 1]))
}
for (i in 201:n) {
  sigma_2[i + 1] <- 1 + 0.1 * x[i]^2 + 0.5 * sigma_2[i]
  x[i + 1] <- rnorm(1, 0, sqrt(sigma_2[i + 1]))
}
result <- suppressWarnings(
  detect_garch(x[-1], c(1, 1), include.mean = FALSE)
)
summary(result)
plot(result)

```

```

set.seed(1)
n <- 120
sigma_2 <- rep(1, n + 1)
x <- rep(0, n + 1)
for (i in seq_len(60)) {
  sigma_2[i + 1] <- 10 + 0.5 * x[i]^2 + 0.3 * sigma_2[i]
  x[i + 1] <- rnorm(1, 0, sqrt(sigma_2[i + 1]))
}
for (i in 61:n) {
  sigma_2[i + 1] <- 0.2 + 0.05 * x[i]^2 + 0.1 * sigma_2[i]
  x[i + 1] <- rnorm(1, 0, sqrt(sigma_2[i + 1]))
}
result <- suppressWarnings(
  detect_garch(x[-1], c(1, 1), include.mean = FALSE)
)
summary(result)
plot(result)

```

```

set.seed(1)
n <- 150
sigma_2 <- rep(1, n + 1)
x <- rep(0, n + 1)
for (i in seq_len(75)) {
  sigma_2[i + 1] <- 10 + 0.5 * x[i]^2 + 0.3 * sigma_2[i]
  x[i + 1] <- rnorm(1, 0, sqrt(sigma_2[i + 1]))
}
for (i in 76:n) {
  sigma_2[i + 1] <- 0.2 + 0.05 * x[i]^2 + 0.1 * sigma_2[i]
  x[i + 1] <- rnorm(1, 0, sqrt(sigma_2[i + 1]))
}
result <- suppressWarnings(

```

```

    detect_garch(x[-1], c(1, 1), include.mean = FALSE)
  )
  summary(result)
  plot(result)

```

detect_kernel

Find change points efficiently via kernel change point detection

Description

[detect_kernel\(\)](#) and [fastcpd.kcp\(\)](#) detect change points in the full distribution of the data using the kernel change point (KCP) approach. The RBF kernel $k(x, y) = \exp(-\|x - y\|^2 / (2\sigma^2))$ is approximated by random Fourier features via Bochner's theorem, and change points in the resulting kernel mean embedding are detected with the existing PELT infrastructure. This is $O(nD)$ in time — compared to $O(n^2)$ for the exact kernel PELT — where D is `order[1]`.

Usage

```
detect_kernel(data, order = c(100L, 0), ...)
```

```
detect_kcp(data, order = c(100L, 0), ...)
```

```
fastcpd_kcp(data, order = c(100L, 0), ...)
```

```
fastcpd.kcp(data, order = c(100L, 0), ...)
```

Arguments

<code>data</code>	A numeric vector or a matrix with one row per observation.
<code>order</code>	A numeric vector of length up to 2: <code>order[1]</code> is the number of random Fourier features (default 100; larger values improve accuracy at the cost of speed) and <code>order[2]</code> is the RBF kernel bandwidth σ (default 0, which triggers the median heuristic $\sigma = \sqrt{\text{median}(d^2)/2}$ on a subsample of up to 1000 observations). Call <code>set.seed()</code> before this function for reproducible results.
<code>...</code>	Other arguments passed to detect() , for example, <code>segment_count</code> .

Value

A [fastcpd](#) object.

See Also

[detect\(\)](#)

Examples

```
set.seed(1)
p_values <- c(runif(200, 0, 0.05), runif(200, 0, 1))
result <- detect_kernel(p_values)
summary(result)

set.seed(7)
identical_result <- detect_kernel(
  rep(3, 20), order = c(8, 0), beta = 2, trim = 0, cp_only = TRUE
)
```

detect_lasso

Find change points efficiently in penalized linear regression models

Description

[detect_lasso\(\)](#) and [fastcpd.lasso\(\)](#) are wrapper functions of [detect\(\)](#) to find change points in penalized linear regression models. The function is similar to [detect\(\)](#) except that the data is by default a matrix or data frame with the response variable as the first column and thus a formula is not required here.

Usage

```
detect_lasso(data, ...)

fastcpd_lasso(data, ...)

fastcpd.lasso(data, ...)
```

Arguments

data	A matrix or a data frame with the response variable as the first column.
...	Other arguments passed to detect() , for example, segment_count.

Value

A [fastcpd](#) object.

See Also

[detect\(\)](#)

Examples

```

if (
  requireNamespace("ggplot2", quietly = TRUE) &&
  requireNamespace("mvtnorm", quietly = TRUE)
) {
  set.seed(1)
  n <- 480
  p_true <- 5
  p <- 50
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_0 <- rbind(
    runif(p_true, -5, -2),
    runif(p_true, -3, 3),
    runif(p_true, 2, 5),
    runif(p_true, -5, 5)
  )
  theta_0 <- cbind(theta_0, matrix(0, ncol = p - p_true, nrow = 4))
  y <- c(
    x[1:80, ] %*% theta_0[1, ] + rnorm(80, 0, 1),
    x[81:200, ] %*% theta_0[2, ] + rnorm(120, 0, 1),
    x[201:320, ] %*% theta_0[3, ] + rnorm(120, 0, 1),
    x[321:n, ] %*% theta_0[4, ] + rnorm(160, 0, 1)
  )
  result <- detect_lasso(
    cbind(y, x),
    multiple_epochs = function(segment_length) if (segment_length < 30) 1 else 0
  )
  summary(result)
  plot(result)

  # Combine estimated thetas with true parameters
  thetas <- result@thetas
  thetas <- cbind.data.frame(thetas, t(theta_0))
  names(thetas) <- c(
    "segment 1", "segment 2", "segment 3", "segment 4",
    "segment 1 truth", "segment 2 truth", "segment 3 truth", "segment 4 truth"
  )
  thetas$coordinate <- c(seq_len(p_true), rep("rest", p - p_true))

  # Melt the data frame using base R (i.e., convert from wide to long format)
  data_cols <- setdiff(names(thetas), "coordinate")
  molten <- data.frame(
    coordinate = rep(thetas$coordinate, times = length(data_cols)),
    variable = rep(data_cols, each = nrow(thetas)),
    value = as.vector(as.matrix(thetas[, data_cols]))
  )

  # Remove the "segment " and " truth" parts to extract the segment number
  molten$segment <- gsub("segment ", "", molten$variable)
  molten$segment <- gsub(" truth", "", molten$segment)

  # Compute height: the numeric value of the segment plus an offset for truth values

```

```

molten$height <- as.numeric(gsub("segment.*", "", molten$segment)) +
  0.2 * as.numeric(grepl("truth", molten$variable))

# Create a parameter indicator based on whether the variable corresponds to truth or estimation
molten$parameter <- ifelse(grepl("truth", molten$variable), "truth", "estimated")

p <- ggplot2::ggplot() +
  ggplot2::geom_point(
    data = molten,
    ggplot2::aes(
      x = value, y = height, shape = coordinate, color = parameter
    ),
    size = 4
  ) +
  ggplot2::ylim(0.8, 4.4) +
  ggplot2::ylab("segment") +
  ggplot2::theme_bw()
print(p)
}

```

detect_lm

Find change points efficiently in linear regression models

Description

`detect_lm()` and `fastcpd.lm()` are wrapper functions of `detect()` to find change points in linear regression models. The function is similar to `detect()` except that the data is by default a matrix or data frame with the response variable as the first column and thus a formula is not required here.

Usage

```
detect_lm(data, ...)
```

```
detect_linear_regression(data, ...)
```

```
fastcpd_lm(data, ...)
```

```
fastcpd.lm(data, ...)
```

Arguments

<code>data</code>	A matrix or a data frame with the response variable as the first column.
<code>...</code>	Other arguments passed to <code>detect()</code> , for example, <code>segment_count</code> .

Value

A `fastcpd` object.

See Also[detect\(\)](#)**Examples**

```

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 300
  p <- 4
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_0 <- rbind(c(1, 3.2, -1, 0), c(-1, -0.5, 2.5, -2), c(0.8, 0, 1, 2))
  y <- c(
    x[1:100, ] %%% theta_0[1, ] + rnorm(100, 0, 3),
    x[101:200, ] %%% theta_0[2, ] + rnorm(100, 0, 3),
    x[201:n, ] %%% theta_0[3, ] + rnorm(100, 0, 3)
  )
  result_lm <- detect_lm(cbind(y, x))
  summary(result_lm)
  plot(result_lm)
}

if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 600
  p <- 4
  d <- 2
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta_1 <- matrix(runif(8, -3, -1), nrow = p)
  theta_2 <- matrix(runif(8, -1, 3), nrow = p)
  y <- rbind(
    x[1:350, ] %%% theta_1 + mvtnorm::rmvnorm(350, rep(0, d), 3 * diag(d)),
    x[351:n, ] %%% theta_2 + mvtnorm::rmvnorm(250, rep(0, d), 3 * diag(d))
  )
  result_mlm <- detect_lm(cbind.data.frame(y = y, x = x), p.response = 2)
  summary(result_mlm)
}

```

detect_mean

*Find change points efficiently in mean change models***Description**

[detect_mean\(\)](#) and [fastcpd.mean\(\)](#) are wrapper functions of [detect\(\)](#) to find the mean change. The function is similar to [detect\(\)](#) except that the data is by default a matrix or data frame or a vector with each row / element as an observation and thus a formula is not required here.

Usage

```
detect_mean(data, ...)

fastcpd_mean(data, ...)

fastcpd.mean(data, ...)
```

Arguments

<code>data</code>	A matrix, a data frame or a vector.
<code>...</code>	Other arguments passed to detect() , for example, <code>segment_count</code> .

Value

A [fastcpd](#) object.

See Also

[detect\(\)](#)

Examples

```
set.seed(1)
data <- matrix(c(
  rnorm(300, mean = 0, sd = 10),
  rnorm(400, mean = 50, sd = 10),
  rnorm(300, mean = 2, sd = 10)
))
system.time(result <- detect_mean(data))
summary(result)
plot(result)
set.seed(1)
p <- 3
data <- rbind(
  matrix(rnorm(p * 3e+5, mean = 0, sd = 10), ncol = p),
  matrix(rnorm(p * 4e+5, mean = 50, sd = 10), ncol = p),
  matrix(rnorm(p * 3e+5, mean = 2, sd = 10), ncol = p)
)
system.time(result <- detect_mean(data, cp_only = TRUE))
summary(result)
```

<code>detect_meanvariance</code>	<i>Find change points efficiently in mean variance change models</i>
----------------------------------	--

Description

[detect_meanvariance\(\)](#), [fastcpd.meanvariance\(\)](#), [fastcpd_mv\(\)](#), [fastcpd.mv\(\)](#) are wrapper functions of [detect\(\)](#) to find the meanvariance change. The function is similar to [detect\(\)](#) except that the data is by default a matrix or data frame or a vector with each row / element as an observation and thus a formula is not required here.

Usage

```

detect_meanvariance(data, ...)

detect_mean_variance(data, ...)

fastcpd_meanvariance(data, ...)

fastcpd.meanvariance(data, ...)

fastcpd_mv(data, ...)

fastcpd.mv(data, ...)

```

Arguments

`data` A matrix, a data frame or a vector.

`...` Other arguments passed to [detect\(\)](#), for example, `segment_count`.

Value

A [fastcpd](#) object.

See Also

[detect\(\)](#)

Examples

```

set.seed(1)
data <- c(
  rnorm(3000, 0, 1),
  rnorm(1000, 10, 1),
  rnorm(3000, 10, 20),
  rnorm(1000, 0, 1)
)
system.time(result <- detect_meanvariance(data))
summary(result)
plot(result)
set.seed(1)
p <- 3
data <- if (requireNamespace("mvtnorm", quietly = TRUE)) {
  rbind(
    mvtnorm::rmvnorm(2e+5, mean = rep(0, p), sigma = diag(1, p)),
    mvtnorm::rmvnorm(1e+5, mean = rep(50, p), sigma = diag(1, p)),
    mvtnorm::rmvnorm(2e+5, mean = rep(0, p), sigma = diag(100, p)),
    mvtnorm::rmvnorm(2e+5, mean = rep(0, p), sigma = diag(1, p)),
    mvtnorm::rmvnorm(1e+5, mean = rep(50, p), sigma = diag(1, p)),
    mvtnorm::rmvnorm(2e+5, mean = rep(50, p), sigma = diag(100, p))
  )
} else {

```

```

    rbind(
      matrix(rnorm(p * 2e+5, mean = 0, sd = 1), ncol = p),
      matrix(rnorm(p * 1e+5, mean = 50, sd = 1), ncol = p),
      matrix(rnorm(p * 2e+5, mean = 0, sd = 10), ncol = p),
      matrix(rnorm(p * 2e+5, mean = 0, sd = 1), ncol = p),
      matrix(rnorm(p * 1e+5, mean = 50, sd = 1), ncol = p),
      matrix(rnorm(p * 2e+5, mean = 50, sd = 10), ncol = p)
    )
  }
  system.time(result <- detect_meanvariance(data))
  summary(result)
  result@thetas[seq_len(p), ]
  lapply(result@thetas[seq_len(p^2) + p, ], function(thetas) matrix(thetas, p))

```

detect_poisson

Find change points efficiently in Poisson regression models

Description

`detect_poisson()` and `fastcpd.poisson()` are wrapper functions of `detect()` to find change points in Poisson regression models. The function is similar to `detect()` except that the data is by default a matrix or data frame with the response variable as the first column and thus a formula is not required here.

Usage

```

detect_poisson(data, ...)

detect_poisson_regression(data, ...)

fastcpd_poisson(data, ...)

fastcpd.poisson(data, ...)

```

Arguments

<code>data</code>	A matrix or a data frame with the response variable as the first column.
<code>...</code>	Other arguments passed to <code>detect()</code> , for example, <code>segment_count</code> .

Value

A `fastcpd` object.

See Also

`detect()`

Examples

```
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 1100
  p <- 3
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  delta <- rnorm(p)
  theta_0 <- c(1, 0.3, -1)
  y <- c(
    rpois(500, exp(x[1:500, ] %*% theta_0)),
    rpois(300, exp(x[501:800, ] %*% (theta_0 + delta))),
    rpois(200, exp(x[801:1000, ] %*% theta_0)),
    rpois(100, exp(x[1001:1100, ] %*% (theta_0 - delta)))
  )
  result <- detect_poisson(cbind(y, x))
  summary(result)
  plot(result)
}
```

detect_quantile

Find change points efficiently in quantile regression models

Description

`detect_quantile()` and `fastcpd.quantile()` are wrapper functions of `detect()` to detect change points in quantile regression models using the pinball (check function) loss $\rho_\tau(u) = u(\tau - \mathbf{1}_{u < 0})$. The function detects changes in the conditional τ -quantile of the response given the covariates. The segment cost is minimised via iteratively reweighted least squares (IRLS).

Usage

```
detect_quantile(data, order = 0.5, ...)
```

```
detect_quantile_regression(data, order = 0.5, ...)
```

```
fastcpd_quantile(data, order = 0.5, ...)
```

```
fastcpd.quantile(data, order = 0.5, ...)
```

Arguments

data	A matrix or a data frame with the response variable as the first column and covariates in the remaining columns.
order	Quantile level τ , a numeric value in (0, 1). The default is 0.5, corresponding to median regression.
...	Other arguments passed to <code>detect()</code> , for example, <code>segment_count</code> .

Value

A `fastcpd` object.

See Also

`detect()`

Examples

```
set.seed(1)
n <- 200
p <- 2
x <- matrix(rnorm(n * p), n, p)
theta_1 <- c(1, -1)
theta_2 <- c(-1, 1)
y <- c(
  x[1:100, ] %%% theta_1 + rnorm(100),
  x[101:200, ] %%% theta_2 + rnorm(100)
)
result <- detect_quantile(cbind(y, x), order = 0.5)
summary(result)
set.seed(42)
n <- 400
x <- rep(1, n)
eps <- rnorm(n)
eps[c(50, 51, 52, 300, 301, 302)] <- -15
y <- c(0 + eps[1:200], 3 + eps[201:400])
dat <- cbind(y, x)
result_lm <- detect_lm(dat)
summary(result_lm)
result_qr <- detect_quantile(dat, order = 0.5)
summary(result_qr)
```

detect_rank

Find change points efficiently via rank-based change point detection

Description

`detect_rank()` and `fastcpd.rank()` detect change points using a rank-based, distribution-free cost. Each column is replaced by its global rank centred at zero, and change points in the mean of these centred ranks are detected with the existing PELT infrastructure. The result is fully deterministic and requires no bandwidth selection. The result reports `family = "rank"` and retains the original input; fitted costs, residuals, parameters, and confidence calculations use the centred ranks in every language. The method is most powerful for location shifts; for scale-only or general distributional changes, `detect_kernel()` is preferable.

Usage

```
detect_rank(data, ...)

fastcpd_rank(data, ...)

fastcpd.rank(data, ...)
```

Arguments

`data` A numeric vector or a matrix with one row per observation.

`...` Other arguments passed to [detect\(\)](#), for example, `segment_count`.

Value

A [fastcpd](#) object.

See Also

[detect\(\)](#)

Examples

```
set.seed(1)
x <- c(rt(200, df = 2), rt(200, df = 2, ncp = 5))
result_mean <- detect_mean(x)
summary(result_mean)
result_rank <- detect_rank(x)
summary(result_rank)
rank_profile <- confint(result_rank, method = "profile", window = 2)
rank_wald <- confint(result_rank, parm = "theta", method = "wald")
```

detect_var

Find change points efficiently in VAR(p) models

Description

[detect_var\(\)](#) and [fastcpd.var\(\)](#) are wrapper functions of [detect\(\)](#) to find change points in VAR(p) models. The function is similar to [detect\(\)](#) except that the data is by default a matrix with row as an observation and thus a formula is not required here. Pass the raw, unlagged series; its response dimension is inferred from the number of columns. The old Python-only pre-lagged `p_response/p.response` form is not part of the portable interface.

Usage

```
detect_var(data, order = 1, ...)

fastcpd_var(data, order = 1, ...)

fastcpd.var(data, order = 1, ...)
```

Arguments

`data` A matrix, a data frame or a time series object.

`order` A positive integer specifying the order of the VAR model; defaults to 1 (VAR(1)).

`...` Other arguments passed to [detect\(\)](#), for example, `segment_count`.

Value

A [fastcpd](#) object.

See Also

[detect\(\)](#)

Examples

```
set.seed(1)
n <- 300
p <- 2
theta_1 <- matrix(c(-0.3, 0.6, -0.5, 0.4, 0.2, 0.2, 0.2, -0.2), nrow = p)
theta_2 <- matrix(c(0.3, -0.4, 0.1, -0.5, -0.5, -0.2, -0.5, 0.2), nrow = p)
x <- matrix(0, n + 2, p)
for (i in 1:200) {
  x[i + 2, ] <- theta_1 %*% c(x[i + 1, ], x[i, ]) + rnorm(p, 0, 1)
}
for (i in 201:n) {
  x[i + 2, ] <- theta_2 %*% c(x[i + 1, ], x[i, ]) + rnorm(p, 0, 1)
}
result <- detect_var(x, 2)
summary(result)
```

detect_variance

Find change points efficiently in variance change models

Description

[detect_variance\(\)](#) and [fastcpd.variance\(\)](#) are wrapper functions of [detect\(\)](#) to find the variance change. The function is similar to [detect\(\)](#) except that the data is by default a matrix or data frame or a vector with each row / element as an observation and thus a formula is not required here.

Usage

```
detect_variance(data, ...)
```

```
fastcpd_variance(data, ...)
```

```
fastcpd.variance(data, ...)
```

Arguments

`data` A matrix, a data frame or a vector.
`...` Other arguments passed to `detect()`, for example, `segment_count`.

Value

A `fastcpd` object.

See Also

`detect()`

Examples

```
set.seed(1)
data <- c(rnorm(300, 0, 1), rnorm(400, 0, 10), rnorm(300, 0, 1))
system.time(result <- detect_variance(data))
summary(result)
plot(result)
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  p <- 3
  data <- rbind(
    mvtnorm::rmvnorm(
      3e+5, rep(0, p), crossprod(matrix(runif(p^2) * 2 - 1, p))
    ),
    mvtnorm::rmvnorm(
      4e+5, rep(0, p), crossprod(matrix(runif(p^2) * 2 - 1, p))
    ),
    mvtnorm::rmvnorm(
      3e+5, rep(0, p), crossprod(matrix(runif(p^2) * 2 - 1, p))
    )
  )
  result_time <- system.time(
    result <- detect_variance(data, cp_only = TRUE)
  )
  print(result_time)
  summary(result)
}
```

estimate_variance

Estimate variance for change point models

Description

Estimate variance for change point models

Usage

```
estimate_variance(data, family = "mean", ...)
```

Arguments

data	Data used for variance estimation.
family	Model family. One of "mean", "median", "linear_regression", "lm", or "arma".
...	Additional arguments passed to the family-specific estimator.

Value

The estimated variance, covariance matrix, or ARMA variance summary.

See Also

[estimate_variance_mean()], [estimate_variance_median()], [estimate_variance_linear_regression()], and [estimate_variance_arma()].

estimate_variance_arma

Variance estimation for ARMA model with change points

Description

Estimate the variance for each block and then take the average.

Usage

```
estimate_variance_arma(data, p, q, max_order = p * q)
```

```
variance_arma(data, p, q, max_order = p * q)
```

```
variance.arma(data, p, q, max_order = p * q)
```

Arguments

data	A one-column matrix or a vector.
p	The order of the autoregressive part.
q	The order of the moving average part.
max_order	The maximum order of the AR model to consider.

Value

A numeric value representing the variance.

Examples

```

set.seed(1)
n <- 300
w <- rnorm(n + 3, 0, 10)
x <- rep(0, n + 3)
for (i in 1:200) {
  x[i + 3] <- 0.1 * x[i + 2] - 0.3 * x[i + 1] + 0.1 * x[i] +
    0.1 * w[i + 2] + 0.5 * w[i + 1] + w[i + 3]
}
for (i in 201:n) {
  x[i + 3] <- 0.3 * x[i + 2] + 0.1 * x[i + 1] - 0.3 * x[i] -
    0.6 * w[i + 2] - 0.1 * w[i + 1] + w[i + 3]
}
(result <- estimate_variance_arma(x[-seq_len(3)], p = 3, q = 2))

```

```
estimate_variance_linear_regression
```

Variance estimation for linear models with change points

Description

Estimate the variance for each block and then take the average.

Usage

```

estimate_variance_linear_regression(
  data,
  d = 1,
  block_size = ncol(data) - d + 1,
  outlier_iqr = Inf
)

```

```

estimate_variance_lm(
  data,
  d = 1,
  block_size = ncol(data) - d + 1,
  outlier_iqr = Inf
)

```

```
variance_lm(data, d = 1, block_size = ncol(data) - d + 1, outlier_iqr = Inf)
```

```
variance.lm(data, d = 1, block_size = ncol(data) - d + 1, outlier_iqr = Inf)
```

Arguments

<code>data</code>	A matrix or a data frame with the response variable as the first column.
<code>d</code>	The dimension of the response variable.

`block_size` The size of the blocks to use for variance estimation.

`outlier_iqr` The number of interquartile ranges to use as a threshold for outlier detection.

Value

A numeric value representing the variance.

Examples

```
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  n <- 300
  p <- 4
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta <- rbind(c(1, 3.2, -1, 0), c(-1, -0.5, 2.5, -2), c(0.8, 0, 1, 2))
  y <- c(
    x[1:100, ] %*% theta[1, ] + rnorm(100, 0, 3),
    x[101:200, ] %*% theta[2, ] + rnorm(100, 0, 3),
    x[201:n, ] %*% theta[3, ] + rnorm(100, 0, 3)
  )
  (sigma2 <- estimate_variance_linear_regression(cbind(y, x)))

  set.seed(1)
  n <- 300
  p <- 4
  d <- 2
  x <- mvtnorm::rmvnorm(n, rep(0, p), diag(p))
  theta <- cbind(c(1, 3.2, -1, 0), c(-1, -0.5, 2.5, -2), c(0.8, 0, 1, 2))
  theta <- cbind(theta, theta)
  y <- rbind(
    x[1:100, ] %*% theta[, 1:2] +
      mvtnorm::rmvnorm(100, rep(0, d), diag(3, d)),
    x[101:200, ] %*% theta[, 3:4] +
      mvtnorm::rmvnorm(100, rep(0, d), diag(3, d)),
    x[201:n, ] %*% theta[, 5:6] +
      mvtnorm::rmvnorm(100, rep(0, d), diag(3, d))
  )
  (sigma <- estimate_variance_linear_regression(cbind(y, x), d = d))
}
```

estimate_variance_mean

Variance estimation for mean change models

Description

Implement Rice estimator for variance in mean change models.

Usage

```
estimate_variance_mean(data)

variance_mean(data)

variance.mean(data)
```

Arguments

data A matrix or a data frame with data points as each row.

Value

A matrix representing the variance-covariance matrix or a numeric value representing the variance.

Examples

```
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  p <- 3
  data <- rbind(
    mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(100, p)),
    mvtnorm::rmvnorm(400, mean = rep(50, p), sigma = diag(100, p)),
    mvtnorm::rmvnorm(300, mean = rep(2, p), sigma = diag(100, p))
  )
  (sigma <- estimate_variance_mean(data))
}
```

estimate_variance_median

Variance estimation for median change models

Description

Implement Rice estimator.

Usage

```
estimate_variance_median(data)

variance_median(data)

variance.median(data)
```

Arguments

data A vector of data points.

Value

A numeric value representing the variance.

Examples

```
(sigma2 <- estimate_variance_median(well_log))
```

fastcpd-class

An S4 class to store the output created with `detect()`

Description

This S4 class stores the output from `detect()` and `detect_family`. A fastcpd object consist of several slots including the call to `detect()`, the data used, the family of the model, the change points, the cost values, the residuals, the estimated parameters and a boolean indicating whether the model was fitted with only change points or with change points and parameters, which you can select using @.

Slots

`call` The call of the function.

`data` The data passed to the function.

`order` The order of the time series model.

`family` The family of the model.

`cp_set` The set of change points.

`raw_cp_set` The untrimmed set of change points returned by the native dynamic-programming traceback.

`cost_values` The cost function values for each segment.

`residuals` The residuals of the model with change points, stored with one row per original observation and one column per response. Leading lag rows are missing for autoregressive models. Used only for built-in families.

`thetas` The estimated parameters for each segment. Used only for built-in families.

`cp_only` A boolean indicating whether `detect()` was run to return only the change points or the change points with the estimated parameters and cost values for each segment.

occupancy	<i>Occupancy Detection Data Set</i>
-----------	-------------------------------------

Description

Data set for binary classification of room occupancy from temperature, humidity, light and CO2 measurements. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute.

Usage

```
occupancy
```

Format

A data frame with 9752 rows and 7 variables:

date Character in the format "YYYY-MM-DD hh:mm:ss" from 2015-02-11 14:48:00 to 2015-02-18 09:19:00

Temperature Temperature in Celsius

Humidity Humidity

Light Light

CO2 CO2

HumidityRatio Humidity Ratio

Occupancy Binary variable with values 0 (unoccupied) and 1

Source

<<https://github.com/LuisM78/Occupancy-detection-data>>

plot.fastcpd	<i>Plot the data and the change points for a fastcpd object</i>
--------------	---

Description

Plot the data and the change points for a [fastcpd](#) object

Usage

```
## S3 method for class 'fastcpd'
plot(
  x,
  color_max_count = Inf,
  data_point_alpha = 0.8,
  data_point_linewidth = 0.5,
  data_point_size = 1,
  legend_position = "none",
  panel_background = ggplot2::element_blank(),
  panel_border = ggplot2::element_rect(fill = NA, colour = "grey20"),
  panel_grid_major = ggplot2::element_line(colour = "grey98"),
  panel_grid_minor = ggplot2::element_line(colour = "grey98"),
  segment_separator_alpha = 0.8,
  segment_separator_color = "grey",
  segment_separator_linetype = "dashed",
  strip_background = ggplot2::element_rect(fill = "grey85", colour = "grey20"),
  xlab = NULL,
  ylab = NULL,
  ...
)

## S4 method for signature 'fastcpd,missing'
plot(
  x,
  color_max_count = Inf,
  data_point_alpha = 0.8,
  data_point_linewidth = 0.5,
  data_point_size = 1,
  legend_position = "none",
  panel_background = ggplot2::element_blank(),
  panel_border = ggplot2::element_rect(fill = NA, colour = "grey20"),
  panel_grid_major = ggplot2::element_line(colour = "grey98"),
  panel_grid_minor = ggplot2::element_line(colour = "grey98"),
  segment_separator_alpha = 0.8,
  segment_separator_color = "grey",
  segment_separator_linetype = "dashed",
  strip_background = ggplot2::element_rect(fill = "grey85", colour = "grey20"),
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

x A [fastcpd](#) object.

color_max_count Maximum number of colors to use for the plotting of segments.

<code>data_point_alpha</code>	Alpha of the data points.
<code>data_point_linewidth</code>	Linewidth of the data points.
<code>data_point_size</code>	Size of the data points.
<code>legend_position</code>	Position of the legend.
<code>panel_background</code>	Background of the panel.
<code>panel_border</code>	Border of the panel.
<code>panel_grid_major</code>	Major grid lines of the panel.
<code>panel_grid_minor</code>	Minor grid lines of the panel.
<code>segment_separator_alpha</code>	Alpha of the segment separator lines.
<code>segment_separator_color</code>	Color of the segment separator lines.
<code>segment_separator_linetype</code>	Linetype of the segment separator lines.
<code>strip_background</code>	Background of the strip.
<code>xlab</code>	Label for the x-axis.
<code>ylab</code>	Label for the y-axis.
<code>...</code>	Ignored.

Value

No return value, called for plotting.

Examples

```
if (requireNamespace("mvtnorm", quietly = TRUE)) {
  set.seed(1)
  p <- 1
  x <- mvtnorm::rmvnorm(300, rep(0, p), diag(p))
  theta_0 <- matrix(c(1, -1, 0.5))
  y <- c(
    x[1:100, ] * theta_0[1, ] + rnorm(100, 0, 1),
    x[101:200, ] * theta_0[2, ] + rnorm(100, 0, 1),
    x[201:300, ] * theta_0[3, ] + rnorm(100, 0, 1)
  )
  result <- detect_lm(cbind(y, x))
  summary(result)
  plot(result)
}
```

plot.fastcpd_confint *Plot confidence intervals for a [fastcpd](#) object*

Description

Visualizes intervals produced by [confint.fastcpd\(\)](#). Change-point intervals are drawn as shaded bands over the data with the point estimates as vertical lines. Parameter intervals are drawn as point ranges for each segment, with one panel per parameter.

Plotting change-point intervals requires the fitted object stored on the fastcpd_confint data frame, so pass the value returned by `confint()` without subsetting it.

Usage

```
## S3 method for class 'fastcpd_confint'
plot(
  x,
  data_point_alpha = 0.8,
  data_point_linewidth = 0.5,
  data_point_size = 1,
  interval_alpha = 0.3,
  interval_color = "steelblue",
  estimate_color = "grey",
  estimate_linetype = "dashed",
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

x	A fastcpd_confint data frame returned by confint.fastcpd() .
data_point_alpha	Alpha of the data points.
data_point_linewidth	Linewidth of the data lines.
data_point_size	Size of the data points.
interval_alpha	Alpha of the confidence interval bands.
interval_color	Color of the confidence intervals.
estimate_color	Color of the point-estimate lines.
estimate_linetype	Linetype of the point-estimate lines.
xlab	Label for the x-axis.
ylab	Label for the y-axis.
...	Ignored.

Value

No return value, called for plotting.

Examples

```
sigma2 <- estimate_variance_median(well_log)
result <- detect_mean(well_log, trim = 0.001, variance_estimation = sigma2)

cp_profile_interval <- confint(
  result,
  parm = "cp",
  method = "profile",
  level = 0.8,
  window = 8
)
plot(cp_profile_interval)

theta_wald_interval <- confint(result, parm = "theta", method = "wald")
plot(theta_wald_interval)
```

```
print.fastcpd
```

Print the call and the change points for a [fastcpd](#) object

Description

Print the call and the change points for a [fastcpd](#) object

Usage

```
## S3 method for class 'fastcpd'
print(x, ...)

## S4 method for signature 'fastcpd'
print(x, ...)
```

Arguments

<code>x</code>	A fastcpd object.
<code>...</code>	Ignored.

Value

Return a (temporarily) invisible copy of the [fastcpd](#) object. Called primarily for printing the change points in the model.

show.fastcpd	Show the available methods for a <i>fastcpd</i> object
--------------	--

Description

Show the available methods for a *fastcpd* object

Usage

```
## S3 method for class 'fastcpd'
show(object)

## S4 method for signature 'fastcpd'
show(object)
```

Arguments

object A *fastcpd* object.

Value

No return value, called for showing a list of available methods for a *fastcpd* object.

summary.fastcpd	Show the summary of a <i>fastcpd</i> object
-----------------	---

Description

Show the summary of a *fastcpd* object

Usage

```
## S3 method for class 'fastcpd'
summary(object, ...)
```

```
## S4 method for signature 'fastcpd'
summary(object, ...)
```

Arguments

object A *fastcpd* object.
 ... Ignored.

Value

Return a (temporarily) invisible copy of the *fastcpd* object. Called primarily for printing the summary of the model including the call, the change points, the cost values and the estimated parameters.

`transcriptome`*Transcription Profiling of 57 Human Bladder Carcinoma Samples*

Description

Transcriptome analysis of 57 bladder carcinomas on Affymetrix HG-U95A and HG-U95Av2 microarrays

Usage`transcriptome`**Format**

A data frame with 2215 rows and 43 variables:

3 Individual 3
4 Individual 4
5 Individual 5
6 Individual 6
7 Individual 7
8 Individual 8
9 Individual 9
10 Individual 10
14 Individual 14
15 Individual 15
16 Individual 16
17 Individual 17
18 Individual 18
19 Individual 19
21 Individual 21
22 Individual 22
24 Individual 24
26 Individual 26
28 Individual 28
30 Individual 30
31 Individual 31
33 Individual 33
34 Individual 34
35 Individual 35

36 Individual 36
37 Individual 37
38 Individual 38
39 Individual 39
40 Individual 40
41 Individual 41
42 Individual 42
43 Individual 43
44 Individual 44
45 Individual 45
46 Individual 46
47 Individual 47
48 Individual 48
49 Individual 49
50 Individual 50
51 Individual 51
53 Individual 53
54 Individual 54
57 Individual 57

Source

<https://www.ebi.ac.uk/biostudies/arrayexpress/studies/E-TABM-147>>
<https://github.com/cran/ecp/tree/master/data>>

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {  
  result <- detect_mean(transcriptome$"10", trim = 0.005)  
  summary(result)  
  plot(result)  
  
  result_all <- detect_mean(  
    transcriptome,  
    beta = (ncol(transcriptome) + 1) * log(nrow(transcriptome)) / 2 * 5,  
    trim = 0  
  )  
  
  plots <- lapply(  
    seq_len(ncol(transcriptome)), function(i) {  
      ggplot2::ggplot(  
        data = data.frame(  
          x = seq_along(transcriptome[, i]), y = transcriptome[, i]  
        ),  
        ggplot2::aes(x = x, y = y)
```

```

    ) +
    ggplot2::geom_line(color = "steelblue") +
    ggplot2::geom_vline(
      xintercept = result_all@cp_set,
      color = "red",
      linetype = "dotted",
      linewidth = 0.5,
      alpha = 0.7
    ) +
    ggplot2::theme_void()
  }
)

if (requireNamespace("gridExtra", quietly = TRUE)) {
  gridExtra::grid.arrange(grobs = plots, ncol = 1, nrow = ncol(transcriptome))
}
}

```

uk_seatbelts

UK Seatbelts Data

Description

Road Casualties in Great Britain 1969–84.

Usage

```
uk_seatbelts
```

Format

uk_seatbelts is a multiple time series, with columns

DriversKilled car drivers killed.

front front-seat passengers killed or seriously injured.

rear rear-seat passengers killed or seriously injured.

kms distance driven.

PetrolPrice petrol price.

VanKilled number of van ('light goods vehicle') drivers.

law 0/1: was the law in effect that month?

Source

R package **datasets**

Examples

```

if (requireNamespace("ggplot2", quietly = TRUE)) {
  result_ar <- detect_ar(diff(uk_seatbelts[, "drivers"], 12), 1, beta = "BIC")
  summary(result_ar)
  plot(result_ar)

  result_lm <- suppressMessages(detect_lm(
    diff(uk_seatbelts[, c("drivers", "kms", "PetrolPrice", "law")], lag = 12)
  ))

  # Compute change point dates:
  cp_dates <- as.POSIXlt(as.Date("1969-01-01", format = "%Y-%m-%d"))
  cp_dates$mon <- cp_dates$mon + (1 + result_lm@cp_set + 12)
  cp_dates <- as.Date(cp_dates)

  # Convert the time series to Date objects:
  # For a monthly ts object, extract year and month manually.
  time_vals <- time(uk_seatbelts)
  years <- floor(time_vals)
  months <- round((time_vals - years) * 12 + 1)
  dates <- as.Date(paste(years, months, "01", sep = "-"), format = "%Y-%m-%d")

  # Prepare the data frame for plotting
  # 'color' is defined similarly to the original code.
  uk_seatbelts_df <- data.frame(
    dates = dates,
    drivers = as.numeric(uk_seatbelts[, "drivers"]),
    color = as.factor((dates < cp_dates[1]) + (dates < cp_dates[2]))
  )

  p <- ggplot2::ggplot(
    data = uk_seatbelts_df,
    ggplot2::aes(x = dates, y = drivers, color = color)
  ) +
    ggplot2::geom_line() +
    ggplot2::geom_vline(
      xintercept = cp_dates,
      linetype = "dashed",
      color = "red"
    ) +
    ggplot2::scale_x_date(date_labels = "%b %Y", date_breaks = "1 year") +
    ggplot2::annotate(
      "text",
      x = cp_dates,
      y = 1025,
      label = format(cp_dates, "%b %Y"),
      color = "blue"
    ) +
    ggplot2::theme_bw() +
    ggplot2::theme(legend.position = "none")
  print(p)
}

```

well_log	<i>Well-log Dataset from Numerical Bayesian Methods Applied to Signal Processing</i>
----------	--

Description

This is the well-known well-log dataset used in many changepoint papers obtained from Alan Turing Institute GitHub repository and licensed under the MIT license.

Usage

```
well_log
```

Format

A Time-Series of length 4050.

Source

<<https://github.com/alan-turing-institute/TCPD>>

Examples

```
result <- detect_mean(well_log, trim = 0.001)
summary(result)
plot(result)

if (requireNamespace("matrixStats", quietly = TRUE)) {
  sigma2 <- estimate_variance_median(well_log)
  median_loss <- function(data) {
    sum(abs(data - matrixStats::colMedians(data))) / sqrt(sigma2) / 2
  }
  result <- detect(
    formula = ~ x - 1,
    data = cbind.data.frame(x = well_log),
    cost = median_loss,
    trim = 0.002
  )
  summary(result)

  segment_starts <- c(1, result@cp_set)
  segment_ends <- c(result@cp_set - 1, length(well_log))
  residual <- NULL
  for (segment_index in seq_along(segment_starts)) {
    segment <-
      well_log[segment_starts[segment_index]:segment_ends[segment_index]]
    residual <- c(residual, segment - median(segment))
  }

  result@residuals <- matrix(residual)
```

```
    result@data <- data.frame(x = c(well_log))  
    plot(result)  
  }
```

Index

* datasets

- bitcoin, [3](#)
- occupancy, [42](#)
- transcriptome, [48](#)
- uk_seatbelts, [50](#)
- well_log, [52](#)

bitcoin, [3](#)

confint.fastcpd, [4](#)
confint.fastcpd(), [45](#)

detect, [6](#)
detect(), [6](#), [8](#), [9](#), [16–22](#), [24](#), [25](#), [27–36](#), [41](#)
detect_ar, [16](#)
detect_ar(), [16](#), [18](#), [22](#)
detect_arima, [17](#)
detect_arima(), [17](#), [22](#)
detect_arma, [18](#)
detect_arma(), [17](#), [18](#), [22](#)
detect_binomial, [20](#)
detect_binomial(), [20](#), [22](#)
detect_exponential, [21](#)
detect_exponential(), [21](#), [22](#)
detect_family, [7](#), [8](#), [11](#), [22](#), [41](#)
detect_garch, [22](#)
detect_garch(), [22](#)
detect_kcp (detect_kernel), [24](#)
detect_kernel, [24](#)
detect_kernel(), [22](#), [24](#), [33](#)
detect_lasso, [25](#)
detect_lasso(), [22](#), [25](#)
detect_linear_regression (detect_lm), [27](#)
detect_lm, [27](#)
detect_lm(), [22](#), [27](#)
detect_logistic_regression
 (detect_binomial), [20](#)
detect_mean, [28](#)
detect_mean(), [22](#), [28](#)

detect_mean_variance
 (detect_meanvariance), [29](#)
detect_meanvariance, [29](#)
detect_meanvariance(), [22](#), [29](#)
detect_poisson, [31](#)
detect_poisson(), [22](#), [31](#)
detect_poisson_regression
 (detect_poisson), [31](#)
detect_quantile, [32](#)
detect_quantile(), [22](#), [32](#)
detect_quantile_regression
 (detect_quantile), [32](#)
detect_rank, [33](#)
detect_rank(), [22](#), [33](#)
detect_var, [34](#)
detect_var(), [22](#), [34](#)
detect_variance, [35](#)
detect_variance(), [22](#), [35](#)

estimate_variance, [36](#)
estimate_variance_arma, [37](#)
estimate_variance_linear_regression,
 [38](#)
estimate_variance_lm
 (estimate_variance_linear_regression),
 [38](#)
estimate_variance_mean, [39](#)
estimate_variance_median, [40](#)

fastcpd, [4](#), [6](#), [11](#), [16](#), [18–22](#), [24](#), [25](#), [27](#),
 [29–31](#), [33–36](#), [42](#), [43](#), [45–47](#)
fastcpd (detect), [6](#)
fastcpd-class, [41](#)
fastcpd.ar (detect_ar), [16](#)
fastcpd.ar(), [16](#)
fastcpd.arima (detect_arima), [17](#)
fastcpd.arima(), [17](#)
fastcpd.arma (detect_arma), [18](#)
fastcpd.arma(), [18](#)
fastcpd.binomial (detect_binomial), [20](#)

`fastcpd.binomial()`, 20
`fastcpd.exponential`
 (`detect_exponential`), 21
`fastcpd.exponential()`, 21
`fastcpd.family` (`detect_family`), 22
`fastcpd.garch` (`detect_garch`), 22
`fastcpd.garch()`, 22
`fastcpd.kcp` (`detect_kernel`), 24
`fastcpd.kcp()`, 24
`fastcpd.lasso` (`detect_lasso`), 25
`fastcpd.lasso()`, 25
`fastcpd.lm` (`detect_lm`), 27
`fastcpd.lm()`, 27
`fastcpd.mean` (`detect_mean`), 28
`fastcpd.mean()`, 28
`fastcpd.meanvariance`
 (`detect_meanvariance`), 29
`fastcpd.meanvariance()`, 29
`fastcpd.mv` (`detect_meanvariance`), 29
`fastcpd.mv()`, 29
`fastcpd.poisson` (`detect_poisson`), 31
`fastcpd.poisson()`, 31
`fastcpd.quantile` (`detect_quantile`), 32
`fastcpd.quantile()`, 32
`fastcpd.rank` (`detect_rank`), 33
`fastcpd.rank()`, 33
`fastcpd.var` (`detect_var`), 34
`fastcpd.var()`, 34
`fastcpd.variance` (`detect_variance`), 35
`fastcpd.variance()`, 35
`fastcpd.ar` (`detect_ar`), 16
`fastcpd.arima` (`detect_arima`), 17
`fastcpd.arma` (`detect_arma`), 18
`fastcpd.binomial` (`detect_binomial`), 20
`fastcpd.exponential`
 (`detect_exponential`), 21
`fastcpd.family` (`detect_family`), 22
`fastcpd.garch` (`detect_garch`), 22
`fastcpd.kcp` (`detect_kernel`), 24
`fastcpd.lasso` (`detect_lasso`), 25
`fastcpd.lm` (`detect_lm`), 27
`fastcpd.mean` (`detect_mean`), 28
`fastcpd.meanvariance`
 (`detect_meanvariance`), 29
`fastcpd.mv` (`detect_meanvariance`), 29
`fastcpd.mv()`, 29
`fastcpd.poisson` (`detect_poisson`), 31
`fastcpd.quantile` (`detect_quantile`), 32
`fastcpd.rank` (`detect_rank`), 33
`fastcpd.var` (`detect_var`), 34
`fastcpd.variance` (`detect_variance`), 35
`lm()`, 7
`occupancy`, 42
`plot`, `fastcpd`, `missing-method`
 (`plot.fastcpd`), 42
`plot.fastcpd`, 42
`plot.fastcpd()`, 11
`plot.fastcpd.confint`, 45
`plot.fastcpd.confint()`, 5
`print`, `fastcpd-method` (`print.fastcpd`), 46
`print.fastcpd`, 46
`show`, `fastcpd-method` (`show.fastcpd`), 47
`show.fastcpd`, 47
`summary`, `fastcpd-method`
 (`summary.fastcpd`), 47
`summary.fastcpd`, 47
`summary.fastcpd()`, 11
`transcriptome`, 48
`uk_seatbelts`, 50
`variance.arma` (`estimate_variance_arma`),
 37
`variance.lm`
 (`estimate_variance_linear_regression`),
 38
`variance.mean` (`estimate_variance_mean`),
 39
`variance.median`
 (`estimate_variance_median`), 40
`variance_arma` (`estimate_variance_arma`),
 37
`variance_lm`
 (`estimate_variance_linear_regression`),
 38
`variance_mean` (`estimate_variance_mean`),
 39
`variance_median`
 (`estimate_variance_median`), 40
`well_log`, 52